

extending the set of script language platform widgets to include the entire existing installed base of browser plug-in applications.

Taking each point separately, with regard to point (1) the examiner states that the AAPA discloses in page 3, lines 5-12 the step of parsing an extension command and inherently a source locator parameter.

Referring to page 3, lines 5-12, the ability provided by a scripting language to write text scripts which define interactive applications through run-time parsing of script files is described.

With regard to point (2), the examiner states the AAPA discloses, at page 2, lines 28-30 and page 3, lines 5-12, providing plug-in extension code for a scripting language platform to fetch objects by an inherent locator parameter.

Referring to page 2, lines 28-30, the Web document parsing model is described which allows late binding of plug-in application components. A web browser parses the text of an HTML document and when an <EMBED> tag is parsed automatically invokes external objects referenced by the tag.

Responding first to point (1), the examiner is correct that parsing of scripts is part of the prior art. However, the parsing code is part of a claimed combination that is not obvious in view of the prior art.

Responding to point (2), as described in the background section of the application and known in the art, the browser code includes code for parsing an HTML document and responding to embedded tags in the document having a particular syntax. When the browser parses an EMBED tag it automatically invokes, for example, an external plug-in application. The browser includes a very specific API for the browser plug-in and the browser plug-in application must be written to conform to a standard architecture to interface with the browser API. As a result, the rich functionally provided by existing browser plug-in applications can only be utilized by browser applications including code for implementing the plug-in API.

On the other hand, scripting languages, such as Tcl, have their own APIs providing extensibility in the form of Widgets. Widget code must be written to be compatible with the widget extension API of the scripting language. For a particular scripting language, the Widgets available are limited to those widgets created by programmers for the particular scripting language. However, as described at page 3, lines 12-14 of the application, the interpreter extension mechanism does not

provide for late binding of component application code with application data, as the browser plug-in does.

The claimed computer program product is plug-in extension code that extends the set of widgets used in the graphical script language platform interface to include web browser plug-in applications. Thus, the universe of existing browser plug-in applications is made available to programmer's writing scripts in a particular scripting language.

To accomplish this, the claim recites fetching program code for, when the extension command is parsed, fetching objects referenced by the source locator parameter using standard internet procedures. The examiner is correct that existing script languages parse the script text. However, there is no disclosure in the AAPA, or any other cited reference, of code in a script language plug-in interface extension code that uses standard internet procedures to fetch objects referenced by a source locator parameter included in the syntax of the extension command. This fetching of objects allows the late binding of the object with the plug-in application code which, as described above, is not possible with the existing script interpreter extension mechanism.

Thus, the extension command allows a script writer to type EMBED tag information into a script and responds to the EMBED tag information by fetching the referenced objects, e.g., an object to be processed by a browser plug-in application.

However, as described above, the browser plug-in application is written to be compatible with a browser program API and is not compatible with the script language extension API. Accordingly, the claim also recites program code in the plug-in-interface extension for causing the computer to allow the plug-in application to display and provide interactive processing of a data and/or program object, referenced by the source locator parameter, within a plug-in-interface extension-controlled window of the graphical script language platform interface. The extension code allows the plug-in application to provide interactive processing in a window in the graphical user interface of the scripting language.

Accordingly, the extension program code itself provides the necessary API to allow the browser plug-in to manipulate object data within the browser plug-in application window controlled by the extension code. Because of this program extension code, from the plug-in's point of view the script language extension appears to be a conventional Web browser.

Further, the claim also recites program code for causing the computer to allow manipulation, under control of the script, of the browser plug-in application so that that the browser-

plug in application can be manipulated as a widget native to the script language platform. Thus, the script language programmer has the ability to manipulate the browser using instructions in the script.

Turning now to point (3) raised by the examiner. The examiner states that the features upon which the applicant relies, i.e., extending the set of script language platform widgets to include the entire existing installed base of browser plug-in applications is not recited in the claims.

However, as is apparent from the above, this feature is a RESULT of the claimed program code structures recited in the claims. The various program code elements automatically invoke the plug-in browser code, provide the necessary interface between the browser plug-in application and the extension API of the scripting language, pass data to the plug-in code, create a scripting language user interface controlled window for the plug-in application to process data, and manipulate the browser plug-in application as a native widget.

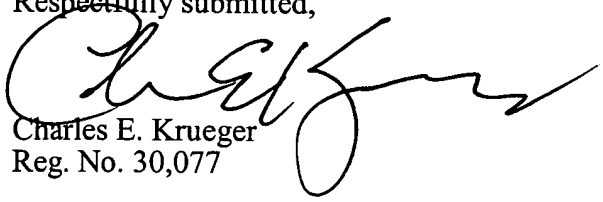
CONCLUSION

In view of the above, it is apparent that the claimed program code elements are not disclosed in the cited references. Further, the novel function of the claimed elements of providing plug-in-interface extension code to provide a browser like API to allow the scripting language interface to employ browser plug-in applications as widget components within the graphical script language program interface is not taught or suggested by the references. Therefore there is no teaching or suggestion in the references that would make the claimed combination obvious to a person of ordinary skill in the art.

Accordingly, Applicants believe all claims now pending in this Application are in condition for allowance. The issuance of a formal Notice of Allowance at an early date is respectfully requested.

If the Examiner believes a telephone conference would expedite prosecution of this application, please telephone the undersigned at (925) 944-3320.

Respectfully submitted,


Charles E. Krueger
Reg. No. 30,077

Doyle :

Application No.: 09/481,984

Page 6

PATENT

Walnut Creek, CA 94596

Tel: (925) 944-3320 / Fax: (925) 944-3363



Doyle
A/N 09/481,984

APPENDIX

RECEIVED
DEC 17 2002
Technology Center 2100

1 4. A computer program product forming an extension of a scripting
2 language platform that extends a [the] set of widgets used in a [the] graphical script
3 language platform interface to include web browser plug-in applications designed to
4 extend the functionality of web browser programs, where an extension command is added
5 to the scripting language and with an [the] extension command syntax including a source
6 locator [locator] parameter, with the computer program product comprising:
7 a computer-readable storage medium for storing plug-in-interface
8 extension program code for extending the functionality of a script interpreter platform by
9 employing web-browser plug-in applications as widget components within the graphical
10 script language platform [program] interface, said plug-in-interface extension program
11 code comprising:
12 parsing program code for causing a computer to parse a
13 script to identify an extension command and the source locator parameter
14 included in the syntax of the extension command;
15 fetching program code for, when the extension command is
16 parsed, fetching objects referenced by the source locator parameter using standard
17 internet procedures;
18 program code for causing the computer to allow the web
19 browser plug-in application to display and provide interactive processing of a data
20 and/or program object, referenced by the source locator parameter, within a plug-
21 in-interface extension-controlled window of the graphical script language
22 platform interface; and
23 program code for causing the computer to allow manipulation, under
24 control of a [the] script, of the browser plug-in application so that [that] the web browser-
25 plug in application can be manipulated as a widget native to the scripting language
26 platform.